# Interfacing to the DØ Data Distributor

Gerald M. Guglielmo[*]
*(FNAL CD/ODS/OSP)*

January 26, 1999
Draft Version 0.2

**Abstract**

This document serves as a basic introduction on how to interface to the DØ Run II Data Distributor. The Data Distributor is responsible for receiving events from the collectors/routers and distributing events according to requests made by Examine client programs. The basic interface at the moment allows for an Examine client to connect to the Data Distributor and request a queue be allocated to handle events for the client. The interface further allows the client to request events, either one at a time or as a group. There is also the ability to request that statistics for the client's queue, or all event queues in the Distributor, be sent.

## 1 Connecting to the Data Distributor

Connecting to the Distributor program involves making a client connection instance and specifying the host and port that the Distributor will be listening on.

### 1.1 Opening a Connection to the Distributor

Opening a client connection to the Distributor can be accomplished with one line of code using DØme. This step involves creating a Client_Connection instance and specifying the host and port of the Distributor server. Once the connection is open, it can be used to register callbacks (section 3.1) for handling incoming messages and for sending outgoing messages to the server.

```
Client_Connection con( options->host());
```

[*]gug@fnal.gov

# 2 Making Requests to the Data Distributor

Creating messages for sending between programs is best done using the "Auto_" prefix versions of the message classes. These "Auto_*" classes are typedefs of the nonprefixed classes that are specified in the name (represented by the "*" character) and use the DØme reference pointer template for keeping track of references to an instance of the class. For example, Auto_DistQueReq_Message is a typedef of DistQueReq_Message with automatic reference pointer counting.

## 2.1 Requesting a Queue

Following the initial connection between the Examine client and the Distributor, there should be a request for a queue to be allocated. This request can be tailored to allow acceptance of more than one trigger type by the queue. The request also specifies globally (*i.e.* for all triggers in the queue) whether or not events are allowed to be overwritten when the queue is full. There is a parameter for specifying the maximum number of events the queue can hold, which will be referred to as the queue depth from now on. Also, on a trigger by trigger basis, the request specifies two prescales (of type double) . The first prescale is for all events of the trigger type and allows events to be written to the queue after being prescaled. The second prescale applies only when the queue is full and prescales the number of events that are overwritten when the queue is full. In other words, when the queue is full the second prescale for the trigger is used to determine whether the event should overwrite an event in the queue or be deleted based on the numbers of events that previously were overwritten and deleted because the queue was full.

There are two basic building blocks for generating a queue requests. These two building blocks are trigger requests and queue requests.

### 2.1.1 Creating a Queue Request

A queue request contains the global parameters for the queue. The global parameters are the queue depth and the overwrite flag. In addition to these global parameters, the queue request contains a list of trigger requests. Creating a queue request begins with a call to instantiate a queue request class. To create a queue request first specify the queue depth, overwrite flag, and number of triggers to be associated with the queue. This can be accomplished be creating an instance of the DistQueReq class (section 5.3).

```
DistQueReq req(int depth,bool overwrite,int numberTrigger);
```

### 2.1.2 Adding Trigger Requests to the Queue Request

A trigger request currently has a stream id and trigger id. At the moment these are used to specify a trigger type that events need to match to be accepted by the queue. The distributor is only using the stream id of events since the toy header does not have anything else that can be used at this time. This will have to be changed when something that resembles a more realistic header is defined. So for now it is important to realize that the stream id of an event will be compared to the trigger id values associated with a queue. The toy level 3 simulator that can be used with the Distributor currently implements a random number generator for the stream id of events. The other two pieces of information are the prescales for this class of events. Both of these need to be specified even though the second one, which prescales the overwriting of events, will be ignored if the queue does not allow overwriting of events. You need to create at least 1 trigger request for the queue. A trigger request specifies the stream id, trigger id and two prescales. Note it is wise to add the same number of triggers as was indicated when the queue request class instance was instantiated (section 2.1.1). One way of adding a group of triggers is to create a pointer to a trigger request object, set the pointer to a new instance of the trigger request class and then add the instance to the queue request instance. This method is well suited for embedding in a loop.

```
DistTrigReq *trig;
trig=new DistTrigReq(int stream, int trigger, double pre1, double pre2);
req.addTrigReq(*trig);
delete trig;
```

Notice that the method for adding triggers takes an object of type DistTrigReq (section 5.6) and not a pointer to an object of DistTrigReq. Therefore one could rewrite the above code as follows:

```
DistTrigReq trig(int stream, int trigger, double pre1, double pre2);
req.addTrigReq(trig);
```

After all the triggers have been added, the next step is creating the request message.

### 2.1.3 Creating a Queue Request Message

Creating a queue request message instance is very easy since all the hard work was done already in generating the queue request (section 2.1.1) and adding the trigger requests (section 2.1.2). Once the request message has been created, it can be easily sent by using the send_message method for the connection (in the example code below con is an instance of the Client_Connection class from section 1.1).

```
Auto_DistQueReq_Message auto_msg( new DistQueReq_Message(req) );
con.send_message( auto_msg );
```

At this point the distributor should be placing events in a queue for you. The next step is requesting events.

## 2.2 Requesting Events

Requesting events involves creating an instance of a DØme message class type designed for this purpose. This message class provides a means for specifying the events desired. The first step is creating the message and specifying how many events are desired. For example, if you have a queue in your program that is three events deep you might want to make an initial request for three events, and subsequent requests for only one event. Next the message can be sent to the distributor using the send_message member function of the connection class. When the distributor receives the request, it will addthe number of events requested to the number of outstanding requests for events it is currently aware of for the queue.

### 2.2.1 Creating a Send Event Message

The generation of a message class instance for requesting events is simple. To create this type of message you only need to know how many events you would like to request. Once an Auto_sendEvent_Message instance (section 5.8) has been created, it can be sent using the send_message method of the connection class.

```
Auto_sendEvent_Message auto_msg( new sendEvent_Message( int
numberEvents) );
con.send_message( auto_msg );
```

You can loop around and request more events as desired. In general, it is a good idea to keep track of how many events you have requested and how many you received.

## 2.3 Requesting Statistics

There is another feature available which allows the client to request the statistics on queue usage from the distributor. Requesting statistics for the queues in the Distributor is very easy. This process involves creating a "send statistics" DØme message specifying how many events are requested, and then sending this message to the distributor. The request can be either for all queues currently managed by the distributor, or just the queue associated with the client.

### 2.3.1 Creating a Send Statistics Message

Once again this is simple to do and only requires that you specify whether you want information on all queues or just your specific queue. Requesting information on all queues is achieved by passing in an integer set to zero. Passing a value of one will request information on all queues currently being managed by the Distributor. Values greater than one or less than zero are currently not defined, but are reserved for future use. To invoke this request one creates an instance of the Auto_sendStatistics_Message class (section 5.9) and sends it to the Distributor.

```
Auto_sendStatistics_Message auto_msg2( new sendStatistics_Message(
int type) );
con.send_message( auto_msg2 );
```

## 3 Handling Messages from the Distributor

Receiving messages is handled by the DØme framework. Access to the messages is achieved by registering a callback, a function to be called when a specific message class is received, to process the message.

### 3.1 Registering Callbacks in DØme

Registering callbacks involves informing the processor for the connection what functions should be called when specific message types are received. These functions can be global or class member functions. The preferred method is to use class member functions.

First step in the preferred method is to create a small class with member functions that can be used for the callbacks. There are at least two methods that should be defined in the class, one for each of the two expected DØme messages. One method should handle the Auto_Event_Message type and the other method should handle Auto_DistFullStats_Message type (section 5.2).

```
int receive_event( Auto_Event_Message msg);
int receive_stats( Auto_DistFullStats_Message msg);
```

Inside these member functions you can use the message methods to extract the data. Note that each message class has a dump method that can be used to dump the information to standard output. Once an instance of the class defining these functions has been created and a client connection has been established, then the registering of the callbacks can be done.

```
//method 3 — preferred method myevent2 event2;
register_callback(con, &myevent2::receive_event, event2);
register_callback(con, &myevent2::receive_stats, event2);
```

That is about all there is to it.

### 3.2 Handling Event Messages

Events are passed around inside of an instance of an Auto_Event_Message class. There are several member methods that allow manipulation and access to the data. For example, there is a method called data() which will return a void* pointer to the data. There is also a function called l3_head() for returning the header information.

### 3.3 Handling Statistics Messages

The statistics information is transmitted from the Distributor to the client using an instance of the Auto_DistFullStats_Message class (section 5.2). Most of the statistics information can be written to standard output by using the dump() method. If access to the statistics data is needed, then there is a member function which will return a class containing all of the statistics information. This member function is called statsRequest() and returns an object of type DistFullStats (section 5.1). With this object the number of queues contained in the report is available, and the information on one of the queues can be extracted. The member function for extracting information on a queue is called getQueStats(int), and requires a queue number between 0 and the number of queues listed. The returned object is an instance of the DistQueStats class (section 5.5).

The DistQueStats class provides a method for returning the number of triggers defined for the queue, and also allows return of an instance of the DistTrigStats class as was done in the case of the queue above. The member function is getTrigStats(int) where a trigger index between 0 and the number of triggers is passed in. All members of the DistTrigStats class (section 5.7) can be directly accessed.

## 4 Include Files Useful for Examine

Here is a list of include files that may be needed.

### 4.1 DØme Include Files

```
#include "d0me/Processor.hpp"
#include "d0me/Connection.hpp"
#include "d0me/Get_Options.hpp"
#include "d0me/Ref_Ptr.hpp"
#include "d0me/Server.hpp"
```

### 4.2 Data Logger Include Files

```
#include "datalogger/Event_Message.hpp"
#include "datalogger/Utils.hpp"
```

### 4.3 Distributor Include Files

```
#include "distributor/DistFullStats.hpp"
#include "distributor/DistFullStats_Message.hpp"
#include "distributor/DistQueReq.hpp"
```

```
#include "distributor/DistQueReq_Message.hpp"
#include "distributor/DistQueStats.hpp"
#include "distributor/DistTrigReq.hpp"
#include "distributor/DistTrigStats.hpp"
#include "distributor/sendEvent_Message.hpp"
#include "distributor/sendStatistics_Message.hpp"
```

## 5 Distributor Interface Classes

This section describes the various classes that can be used to interface to the Distributor program with an examine client. There are two basic types of interfaces classes available. The first type of interface class is for making requests of the Distributor and the second type is for accessing information received from the Distributor.

### 5.1 DistFullStats

The DistFullStats class is allows access to information on the statistics for queues being managed by the data distributor. There is also information in the characteristics of the queues. For details on what information is stored, see the section describing the DistQueStats class (section 5.5).

#### 5.1.1 Public Data Members

**vector<DistQueStats> _queue** There is only one public data member in this class called "_queue", which is a vector of DistQueStats objects. Accessing the queue vector is best done through the member functions. The vector starts at index 0.

#### 5.1.2 Public Member Functions

**void add_queue(const DistQueStats&)** This member function adds information for a queue to the class by adding a DistQueStats object to the vector _queue.

**DistFullStats()** Class constructor taking no arguments.

**DistFullStats(int)** Class constructor taking one argument indicating a minimum number of queues that will have information stored in the class.

**DistFullStats(DistFullStats&)** Class copy constructor.

**DistQueStats& getQueStats(int) cont** This member function takes one argument indicating an index of the _queue vector and returns a DistQueStats object containing information on one queue. The function will not change member data.

**int numberQueues() const** This member function returns the number of queues that described in the class. The return value indicates the number of entries in the _queue vector.The function will not change member data.

**size_t length() const** This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

**void dump() const** This function dumps to standard output information on all queues contained in the vector _queue. The function will not change member data.

## 5.2 DistFullStats_Message

The DistFullStats_Message class is a DØ client-server (DØme) message class for transmitting over the network statistics information on queues managed by the Distributor. This is a message class that basically wraps a DistFullStats object.

### 5.2.1 Public Data Members

The only public data member for this class is the message id value.

**static const int MSG_ID** The message id for the DistFullStats_Message class.

### 5.2.2 Public Member Functions

**int msg_id() const** Returns the MSG_ID for DistFullStats_Message message type. The function will not change member data.

**DistFullStats_Message()** Class constructor taking no arguments.

**DistFullStats_Message(const DistFullStats&)** Class constructor taking data object.

**size_t length() const** This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

**DistFullStats& statsRequest()** Returns a DistFullStats object containing the queue information that is contained in the message.

**size_t in(void*,size_t)** This function is used by the client-server framework to decode the message data from XDR format. The first argument is a pointer to the encoded data and the second argument is the size of the data. The return value indicates the size of data decoded and should equal input size if function was successfull.

**size_t out(void*,size_t)** This function is used by the client-server framework to encode the message data to XDR format. The first argument is a pointer to where the encoded data should be stored and the second argument is the size of the data. The return value indicates the size of data encoded and should equal input size if function was successfull.

**void dump() const** This function dumps to standard output information contained in the DistFullStats object in the message. The function will not change member data.

#### 5.2.3 Automatic Reference Counting

There is a typedef that will automatically reference count for an object of this message class. This provides a convenient interface for messages of this class.

**Auto_DistFullStats_Message** is a typedef to use the client-server automatic reference counting.

## 5.3 DistQueReq

This class specifies the parameters for configuring a queue in the Distributor. The class contains the global queue parameters plus a list of trigger specific parameters for each trigger that the queue will be configured to accept.

#### 5.3.1 Public Data Members

There are no public data members for this class.

#### 5.3.2 Public Member Functions

**DistQueReq(int,bool,int)** Class constructor taking 3 arguments. The first argument specifies the maximum depth for the queue. The second argument indicates whether or not the queue will allow events to be overwritten. The third argument states the minimum number of triggers that will be described in the request class (this option may be removed in the future).

**int maxQueueDepth() const** Returns the maximum queue depth allowed for the queue being requested. The function will not change member data.

**void maxQueueDepthSet(int)** The function allows the requested maximum queue depth to be set to a different value in the request.

**bool ovrt() cont** Returns true if the request is for a queue that will allow overwriting of events. The function will not change member data.

**void ovrtSet()** This function can change the value of the overwrite flag in the request object.

**void addtrigger(DistTrigReq&)** This functions adds the parameters for a trigger to the queue request.

**int numberTrigs() const** Returns the number of triggers configured in the queue request. The function will not change member data.

**DistTrigReq& getTrigReq(int) const** This function returns an object containing the trigger parameters for one trigger type in the request. The argument specifies an index, beginning at 0 and an upper limit less than the number of trigger specified. The function will not change member data.

**void removeTrigReqs()** This function removes all trigger requests from the the queue request object.

**size_t length() const** This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

**void dump() const** This function dumps to standard output information about the queue request. The function will not change member data.

## 5.4 DistQueReq_Message

### 5.4.1 Public Data Members

The only public data member for this class is the message id value.

**static const int MSG_ID** The message id for the DistFullStats_Message class.

### 5.4.2 Public Member Functions

**int msg_id() const** Returns the MSG_ID for DistFullStats_Message message type. The function will not change member data.

**DistQueReq_Message()** Class constructor taking no arguments.

**DistQueReq_Message(const DistQueReq&)** Class constructor taking data object.

**size_t length() const** This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

**DistQueReq& QueueRequest()** Returns a DistQueReq object containing the queue request information that is contained in the message.

**size_t in(void*,size_t)** This function is used by the client-server framework to decode the message data from XDR format. The first argument is a pointer to the encoded data and the second argument is the size of the data. The return value indicates the size of data decoded and should equal input size if function was successfull.

**size_t out(void*,size_t)** This function is used by the client-server framework to encode the message data to XDR format. The first argument is a pointer to where the encoded data should be stored and the second argument is the size of the data. The return value indicates the size of data encoded and should equal input size if function was successfull.

**void dump() const** This function dumps to standard output information contained in the DistQueReq object in the message. The function will not change member data.

#### 5.4.3 Automatic Reference Counting

There is a typedef that will automatically reference count for an object of this message class. This provides a convenient interface for messages of this class.

**Auto_DistQueReq_Message** is a typedef to use the client-server automatic reference counting.

## 5.5 DistQueStats

The DistQueStats class contains information on the statistics for a queue reported by the Distributor.

#### 5.5.1 Public Data Members

**int _maxQueueDepth** Indicates the maximum number of events that can be stored in the queue.

**int _sendEventRequested** The number of events requested that have not yet been sent.

**bool _ovrt_flag** True if the queue allows over-writing of events.

**Vector<DistTrigStats> _trig** A vector of objects of class DistTrigStats which hold statistics information for a trigger in the queue maintained by the Distributor.

#### 5.5.2 Public Member Functions

**DistQueStats()** Class constructor taking no arguments.

**DistQueStats(int,int,bool,int)** Class constructor which takes four arguments. The first argument is the queue depth. The second argument is the number of events requested that have not been sent yet. The third argument specifies whether or not the queue allows the over-writing of events. The fourth argument indicates the minimum number of triggers that will have statistics information added to the object.

**DistQueStats(const DistQueStats&)** The class copy constructor.

**void addTrigger(const DistTrigStats&)** This function adds statistics information for a trigger in the queue to the class object.

**int numberTrigs() const** Returns the number of triggers that are described by the object. The function will not change member data.

**size_t length() const** This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

**void dump() const** This function dumps to standard output information contained in the DistQueStats object. The function will not change member data.

## 5.6 DistTrigReq

The DistTrigReq class is used in a request to the Distributor for specifying the parameters for a specific trigger that will be accepted by the queue being requested.

### 5.6.1 Public Data Members

There are no public data members in this class.

### 5.6.2 Public Member Functions

**DistTrigReq(int,int,double,double)** Class constructor taking four arguments. The first argument is the stream id accepted by the queue and associated with this trigger. The second argument is the trigger id. The third argument is a prescale value that will be applied to all events matching the trigger (or stream) id. The fourth argument is a prescale that only applies if event over-writing is allowed for the queue. If over-writing is allowed, then the fourth argument specifies a prescale that applies when the queue is full and allows prescaling of the over-writing of events.

**int stream_id() const** Returns stream id associated with this trigger request. The function will not change member data.

**int trigger_id() const** Returns trigger id associated with this trigger request. The function will not change member data.

**double prescale_all() const** Returns prescale value for all events seen by the queue of the given trigger or stream type. The function will not change member data.

**double prescale_bfull() const** Returns prescale value for when the queue is full. The function will not change member data.

**void dump() const** This function dumps to standard output information contained in the DistTrigReq object. The function will not change member data.

## 5.7 DistTrigStats

The DistTrigStats class contains information on the statistics for a trigger request in a queue reported by the Distributor.

### 5.7.1 Public Data Members

**int _stream_id** The stream id associated with this trigger request being reported.

**int _trigger_id** The trigger id associated with this trigger request being reported.

**double _prescale_all** The overall prescale applied to all events seen by the queue that match the trigger or stream id.

**double _prescale_bfull** The prescale value applied when the buffer is full. The prescale uses the information on number of events accepted and number over-written for the trigger associated trigger or stream type.

**unsigned int _ev_count** Number of events that match the trigger or stream id that have been seen by the queue.

**double _kb_count** The total size of events that match the trigger or stream id that have been seen by the queue.

**unsigned int _ev_del_bfull** Number of events that match the trigger or stream id that have been deleted because the queue was full.

**double _kb_del_bfull** The total size of events that match the trigger or stream id that have been deleted because the queue was full.

**unsigned int _ev_del_prescale** Number of events that match the trigger or stream id that have been deleted because of a prescale.

**double _kb_del_prescale** The total size of events that match the trigger or stream id that have been deleted because of a prescale.

**unsigned int _ev_ovrt_bfull** Number of events that match the trigger or stream id that have been over-written because the queue was full.

**double _kb_ovrt_bfull** The total size of events that match the trigger or stream id that have been over-written because the queue was full.

**unsigned int _ev_depth** Number of events that match the trigger or stream id currently in the queue.

**double _kb_depth** The total size of events that match the trigger or stream id currently in the queue.

**unsigned int _ev_sent** Number of events that match the trigger or stream id that have been sent to the client.

**double _kb_sent** The total size of events that match the trigger or stream id that have been sent to the client.

**double _calc_prescale_all** The value calculated for the overall prescale based on the statistics of the queue for the trigger or stream id associated with the trigger request.

**double _calc_prescale_bfull** The value calculated for the prescale applied when the queue is full based on the statistics of the queue for the trigger or stream id associated with the trigger request.

### 5.7.2 Public Member Functions

**DistTrigStats()** Class constructor taking no arguments.

**void dump() const** This function dumps to standard output information contained in the DistTrigStats object. The function will not change member data.

## 5.8 sendEvent_Message

The sendEvent_Message is a client-sever message class that allows the client to request the Distributor to send events from the queue it is managing for the client. The request allows a variable number of events to be requested at one time.

### 5.8.1 Public Data Members

**static const int MSG_ID** The message id for the sendEvent_Message class.

### 5.8.2 Public Member Functions

**sendEvent_Message()** Class constructor taking no arguments.

**sendEvent_Message(const int)** Class constructor taking one argument specifying the number of events requested.

**int msg_id() const** Returns the MSG_ID for DistFullStats_Message message type. The function will not change member data.

**size_t length() const** This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

**int numberEvents()** Returns the number of events requested to be sent.

**size_t in(void*,size_t)** This function is used by the client-server framework to decode the message data from XDR format. The first argument is a pointer to the encoded data and the second argument is the size of the data. The return value indicates the size of data decoded and should equal input size if function was successfull.

**size_t out(void*,size_t)** This function is used by the client-server framework to encode the message data to XDR format. The first argument is a pointer to where the encoded data should be stored and the second argument is the size of the data. The return value indicates the size of data encoded and should equal input size if function was successfull.

**void dump() const** This function dumps to standard output information contained in the message object. The function will not change member data.

### 5.8.3 Automatic Reference Counting

There is a typedef that will automatically reference count for an object of this message class. This provides a convenient interface for messages of this class.

**Auto_sendEvent_Message** is a typedef to use the client-server automatic reference counting.

## 5.9 sendStatistics_Message

The sendStatistics_Message class allows a client to request that the Distributor send statistics information for either the queue associated with the connection, or all queues maintained by the Distributor.

### 5.9.1 Public Data Members

**static const int MSG_ID** The message id for the sendStatistics_Message class.

### 5.9.2 Public Member Functions

**sendStatistics_Message()** Class constructor taking no arguments.

**sendStatistics_Message(const int)** Class constructor taking one argument which specifies whether only the queue associated with the connection should be reported on (a value of 0) or all queues maintained by the Distributor should be reported on (a value of 1).

**int msg_id() const** Returns the MSG_ID for DistFullStats_Message message type. The function will not change member data.

**size_t length() const** This member function returns the size of memory needed to XDR encode the information for transmitting over the network. The function will not change member data.

**int requestType()** Returns 0 if the request is only for the queue associated with the connection, or 1 if the request is for all queues.

**size_t in(void*,size_t)** This function is used by the client-server framework to decode the message data from XDR format. The first argument is a pointer to the encoded data and the second argument is the size of the data. The return value indicates the size of data decoded and should equal input size if function was successfull.

**size_t out(void*,size_t)** This function is used by the client-server framework to encode the message data to XDR format. The first argument is a pointer to where the encoded data should be stored and the second argument is the size of the data. The return value indicates the size of data encoded and should equal input size if function was successfull.

**void dump() const** This function dumps to standard output information contained in the message object. The function will not change member data.

#### 5.9.3 Automatic Reference Counting

There is a typedef that will automatically reference count for an object of this message class. This provides a convenient interface for messages of this class.

**Auto_sendStatistics_Message** is a typedef to use the client-server automatic reference counting.